Productivity Engineering in the UNIX† Environment

The Berkeley UNIX Consultant Project

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

Presented at Second International G.I. Conference, Munich, October 20-21, 1987

The Berkeley UNIX Consultant Project*

Robert Wilensky

Division of Computer Science
Department of EECS
University of California, Berkeley
Berkeley, Ca. 94720

1. Introduction

Several years ago, we began a project called UC (UNIX Consultant). UC was to function as an intelligent natural language interface that would allow naive users to learn about the UNIX† operating system by interacting with the consultant in ordinary English. We sometimes refer to UC as "an intelligent 'help' facility" to emphasize our intention to construct a consultation system, rather than a natural language front end to an operating system. Whereas front-ends generally take the place of other interfaces, UC was intended to help the user learn how to use an existing one.

We had two major motivations for choosing this task. These can be summarized by saying that we believed the task to be both interesting and doable. It seemed to us that much natural language work, indeed, much of AI research, has fallen into two largely non-intersecting categories: On one hand, there are quite interesting and ambitious projects that have been more the fertile source of exciting speculations than of useful technology. In contrast, there are projects whose scope is severely limited, either to some intrinsically bounded real world task or laboratory micro-world. These projects result in much excitement by the production of a "working system" or successful technology. But such projects have rarely produced much in the way of progress on fundamental issues that comprise the central goals of AI researchers.

Our hope was that the consultation task would requires us to address fundamental problems in natural language processing, planning and problem solving, and knowledge representation, all of which are of interest to us. We believe this to be the case because (1) the domain of an operating system is quite large and complex, (2) users' conceptions of computer systems are often based on other domains, particularly space and containment, and (3) the structure of a consultation session requires the consultant to understand the user's language, hypothesize his intentions, reason about the user's problem, access knowledge about the topic in question, and formulation a reasonable response. In sum, virtually all the problems of language processing and reasoning arise in some fashion.

While the task is interesting, it is nevertheless limited. Arbitrary knowledge of the world is generally not required, as it may be in other natural language tasks, such as text processing. Even knowledge about the domain might be limited in ways that do not compromise the overall integrity of the system. Since the system is a 'help' facility, it need not be capable of handling

*This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA order No. 4871, monitored by Space and Naval Warfare Systems Command under contract N00039-84-C-0089 and by the Office of Naval Research under contract N00014-80-C-0732.
†UNIX is trademark of Bell Laboratories

every task put to it in order to serve a useful function. This is probably less true of systems that are intended to be interfaces. In their case, failure to process a request by the user correctly leaves the user with little recourse. However, a consultant may be quite useful even if it cannot help all the time.

Similarly, there are strategies that might be employed in a consultant task that further reduce the degree of coverage required by the system. For example, if asked a very specific question, it is not unreasonable that a consultant respond by telling the user where to look for the information. Thus the degree of expertise of the consultation system may be circumscribed.

We did not feel that it was necessary or appropriate to produce a product that could actually be used in a real-world setting in order for our system to be considered a success. However, we did feel we should show that one could develop such a system along the lines our research suggested. This would be accomplished by developing an extensible prototype.

2. UC Old and New

We initially built a prototype version of UC consisting largely of "off the shelf" components (Wilensky, Arens, and Chin 1984). While this system seemed to suggest that our goal was feasible, it was deficient in many ways. There were whole components that needed to be included that were not. For example, the initial system made few inferences, and was not capable of planning its own actions. In addition, each individual component was in need of much refinement.

Probably the most important deficiency was in the area of knowledge representation. The initial prototype of UC was implemented in PEARL (Deering, Faletti, and Wilensky 1981). PEARL is an AI language and data base management package that supports frame-like structures similar to those employed by other representation languages, with perhaps some more attention given to efficient retrieval. However, we found that our underlying representational system was inadequate. Unfortunately, the problems with our system were not unique to it, but shared by most other efforts to represent and organize knowledge.

Much of the focus of our recent work has been to address and rectify these problems of knowledge representation. Our critiques of existing knowledge representation schemes, along with our new prescription for these deficiencies, can be found in Wilensky (1986). That report contains a description of KODIAK, the knowledge representation system that our work has led us to, and upon which our current implementation of the UNIX Consultant is based.

Since one's knowledge representation is generally fundamental to the structure of most of the modules of one's systems, developing a new one means redesigning each component around a new representational system. This report is brief description of this new prototype. A more complete description can be found in Wilensky et al. (1986).

3. The Structure of UC

One of the salient characteristics of our approach to natural language processing is to reject the idea of an interface per se. That is, rather than think of a natural language processing program as something to move information through, we conceive of such a program as an intelligent agent that can reason about the user's request.

There are a number of ways in which this approach manifests itself in our work, some familiar and some more novel. For example, there is a considerable amount of inference that goes on in order to interpret the meaning of a user request, given

background knowledge of the domain and what we have thus far inferred about the user. This sort of processing involves knowledge-based inference, user modeling, and speech act-type goal and plan recognition.

Above and beyond this sort of reasoning, the system itself is meant to be an agent with goals of its own. Usually, the system will simply adapt the goals of the user, but in principle, it may not. For example, the user may be laboring under some misconception, in which case it is better to correct his misconception than implement a request based on it; the user's lack of knowledge may cause him to request something which a more knowledgeable user would not, in which case it is better to inform the user of this situation; the user may make some request that violates some other goal of the system, such as preserving the integrity of someone else's work, in which case the user's request may simply be refused. In all such cases, we want the system to reason about the user's request much in the same way that a human consultant might when put in a similar situation.

As a result, UC is probably a more complex system than more conventional natural language front ends. To implement all this functionality, the current version of UC involves a number of components, which are invoked in a more or less serial fashion. Since each of these components is in and of itself a rather major AI undertaking, they cannot be adequately described herein. The following is a cursory overview which conveys the general distribution of labor:

(1) Language Analysis (ALANA)

Language analysis is that component of the understanding process that computes a representation of the content of an utterance. ALANA, written by Charles Cox and extended by Dan Jurafsky, produces a KODIAK representation of the content of an utterance. This representation generally contains only what can be determined from the words and linguistic structures present in the utterance.

We call such an analysis of an utterance its *primal content*. The concept of primal content is related to what is usually described as the *literal meaning* or *sentence meaning* of an utterance. However, unlike literal meaning, the primal content of an utterance involves certain idiomatic interpretations (i. e., it is not necessarily composed from words and general grammatical constructions). Also, the primal content of an utterance may be rather abstract, perhaps so much so that it may not be a suitable candidate for a meaning. For example, the literal meaning of "The cat is on the mat" is generally taken to be a rather conventional situation in which a cat is resting upon a mat. However, the primal content of this sentence would be more abstract, where the contribution of "on" is identical to that in the primal content of "The light fixture is on the ceiling" or "The notice is on the bulletin board". Presumably, this conveys some sort of support relation. Note that such an abstract content appears never to be in itself the meaning of such an utterance.

In contrast to primal content is the *actual content* of an utterance. The actual content is context-dependent, generally requires some amount of inference based on world knowledge, and is a suitable candidate for the meaning of an utterance. For example, the actual content of "The cat is on the mat", without a further context specified, is what the literal meaning of this sentence is generally taken to be. Computing this content from the primal content requires pragmatic knowledge about the kind of support relation a cat and mat are likely to be in, and requires making an inference that cannot not be justified by the meanings of the terms and the grammatical constructions present in the utterance.

(2) Inference (Concretion Mechanism)

The particular kind of inference needed to go from a primal content to an actual content sometimes involves a process known as *concretion* (Wilensky 1983). Concretion is the process of inferring a more specific interpretation of an utterance than is

justified by language alone. Concretion may involve finding a more specific default interpretation, or some other interpretation based on the context. For example, in the "cat is on the mat" example above, the actual content computed is essentially the default support relation between a cat and a mat. In some compelling context, a quite different actual content may be computed from the same primal content.

(There are other possible relations between primal and actual content besides the latter being a more specific interpretation of the former. For example, a conventionalized metaphor might have a primal content that more closely resembles its literal interpretation, but an actual content resembling its metaphoric interpretation. Thus, one analysis of sentences like "John gave Mary a kiss" will have as its primal content an instance of giving, but as its actual content an instance of kissing. We will not pursue further the details of the primal/actual content distinction here. This is largely because, in UC, the need for concretion is widespread, and our handling of other kinds of primal/actual content computations is more haphazard).

In UC, concretion is needed primarily because we need to organize knowledge about more specific interpretations of utterances than can be arrived at through linguistic knowledge alone. For example, if UC is asked the question "How can I delete a file?", ALANA can represent that this is a question about how to delete a file. But it would not have any reason to assume that deleting a file is a specific kind of deleting. Determining that this is so is likely to be important for a number of reasons. For example, knowledge about how to delete a file will be found associated with the concept of "file deletion", say, but not with the concept of deletion in general. Thus UC must infer that "deleting a file" refers to the specific kind of deletion having to do with computer storage in order to perform subsequent tasks like finding plans for accomplishing the user's request.

In UC, concretion is the function of special mechanism designed specifically for that purpose by Dekai Wu. The output of the concretion mechanism is another KODIAK representation, generally one containing more specific concepts than that produce by ALANA.

(3) Goal Analysis (PAGAN)

Having computed an actual content for an utterance, UC then tries to hypothesize the plans and goals under which the user is operating. This level of analysis is performed by PAGAN (Plan And Goal ANalyzer), written by James Mayfield. PAGAN performs a sort of "speak act" analysis of the utterance. The result of this analysis is a KODIAK representation of the network of plans and goals the user is using with respect to UC.

Goal analysis is important in many ways for UC. As is generally well-known, an analysis of this sort is necessary to interpret indirect speech acts, such as "Do you know how to delete a file", or "Could you tell me how to delete a file?" Furthermore, goal analysis helps to provide better answers to questions such as "Does ls -r recursively list subdirectories?" An accurate response to the literal question might simply be "no". But a better response is "No, it reverse the order of the sort of the directory listing; ls -R recursively lists subdirectories". To produce such a response, one needs to realize that the goal underlying the asking of this question is either to find out what ls -r does, or to find out how to recursively list subdirectories. It is the job of the goal analyzer to recognize that such goals are likely to be behind such a question.

(4) Agent (UC Ego)

Having hypothesized what the user wants of the UNIX Consultant, UC must now decide what its own goals should be. Generally, we would expect a system like UC to do what the user requested. But this is not always appropriate. For example, if the user asked how to crash the system, it would be inappropriate for a consultant to give the user the superuser password in order to help. This is inappropriate because a consultant probably has other goals, such as maintaining the integrity of the

system.

To deal with such situations, UC constructed as an agent. This agent reacts to users' requests by forming goals and acting upon them. The central mechanism of UC is called UC Ego, and has been developed by David Chin.

In a typical transaction, UC Ego will adopt the goal of helping the user by finding out what the user wants to know, and then telling it to the user. As the example above illustrates, UC Ego must also detect conflicts between such goals and other goals it may have. As another example of such an interaction, UC also attempts to be educational. Thus, if the user asks UC to actually perform some request, such as telling the user who is on the system, UC should decide to tell the user *how* to perform such a function himself, rather than do what the user requested. UC needs to have some notion of its own goals in order to decide how to best to perform some action other than what the user requested.

UC Ego is one important way in which UC differs from systems designed to be interfaces. While interfaces are generally thought of as passive conduits through which information flows, UC is best thought of as an agent. The agent listens to the user, and is generally helpful. But the agent has its own agenda, and the requests of the user are merely a source of input to the agent.

(5) User Model

A number of components of UC may need some information about the user in order to make an effective choice. For example, an expert user certainly knows how to delete a file. Thus, such a user uttering "Do you know how to delete a file?" is unlikely to be asking for this information – more likely he is testing the consultant's knowledge.

Assessing the knowledge state of the user is the function of the user model. The user model is built up by UC Ego, primarily because they were designed by the same individual. It is exploited by a number of components, including the Expression Mechanism described below.

(6) Domain Planner (UC Planner)

Typically, UC Ego tries to help the user. This usually requires determining a fact that the user would like to know. This task is accomplished by UC Planner. UC Planner is a "domain planner" developed by Marc Luria. While UC Ego infers its own goals, and plans to act on them, UC Planner is given a task by UC Ego of determining what the user wants to accomplish. UC Planner tries to determine how to accomplish this task, using knowledge about UNIX and knowledge about the user's likely goals. UC Planner returns a plan, represented in KODIAK.

(7) Expression Mechanism (UC Express)

Having gotten UC Planner to compute a plan for the user's request, UC Ego now tries to communicate this plan to the user. To do so, it must determine which aspect of the plan are worthy of communication, and how best to communicate them. For example, if it is likely that the user knows how to use commands in general, it might be sufficient just to specify the name of the command. In contrast, it might be helpful to illustrate a general command with a specific example.

UC Express is an "expression mechanism" written by David Chin. Basically, it edits out those parts of the conceptual answer returned by UC Planner that, for some reason, it appears unnecessary to communicate. UC Express may also choose to illustrate an answer in a number of different formats. For example it might illustrate a general answer by generating a

specific example.

The result of UC Express is an annotated KODIAK network, where the annotation specifies which part of the network to be generated.

(8) Language Production (UC Gen)

Once UC has decided what to communicate, it has to put it into words. This is done by a generation program called UC Gen. UC Gen is a simple generator, programmed by Anthony Albert and Marc Luria. It takes the marked KODIAK network produced by UC Express, and using knowledge of English, produces sentences to complete the transaction with the user.

(9) Learning Mechanism (UC Teacher)

Since it is intended that UC be an extensible system, a mechanism has been developed to add new knowledge to the system by talking to it in natural language. This mechanism, called UC Teacher, is the work of James Martin. UC Teacher has capabilities to extend both UC's knowledge base of UNIX facts, as well as its knowledge of English vocabulary.

Note that several UC components deal with goals and plans, but in rather different ways. To minimize confusion, we emphasize the different tasks that these programs perform. PAGAN hypothesizes goals of the user with respect to UC; UCEgo reacts to the user by forming goals of its own and then trying to accomplish them; UCPlanner is given the goal of solving some underlying UNIX problem. Thus PAGAN is a goal analyzer, that is, a program that attempts to determine someone else's goals, while UCEgo and UCPlanner are planners, i. e., programs that determine and produce plans for their own goals.

UCEgo and UCPlanner exist as two separate programs primarily for pragmatic reasons. It may be possible that a single planner could accomplish the tasks of both UCEgo and UCPlanner. Such an integration will have to await a revision of our system.

3.1. UC Examples

In the examples below, each sentence preceded by the prompt "User:" or "Usern:" is supplied by the user. The rest of the text is a computer generated response.

The first set of interactions illustrates some of the more agent-like abilities of UC. In the first example, the User1 is judged to be a novice, since the systems can infer from the previous question that the user does not know how to delete a file. Since this is the case, the system answers the second question without presuming that the user knows the command "who". In the second transaction, a different user, User2, is judged to be at least an intermediate level user, since the first question indicates that he is aware of the existence of inodes, a rather esoteric aspect of UNIX. In response to the second question, which is identical to the second question asked by User1, the system here responds differently. In particular, is presumes that, as a reasonably knowledgeable user, User2 would know about the command "who", and thus uses this presumption to generate a response.

User1:How can I delete a file?

UC: Use rm.
For example to delete the file foo, type 'rm foo'.
User1:What does rwho do?
UC: Rwho is used to list all users on the network, the users' tty, the users' login time, and the users' idle time.

User2:How can i find out the inode of a file?
UC: Type 'ls -i'.
User2:What does rwho do?
UC: Rwho is like who, except rwho lists all users on the network.

User1 (novice) gets explanation of rm's format with an example. User2 (intermediate) already knows ls's format

The following simpler examples illustrate the scope of the current UC prototype. They by no means exhaust the capabilities of the system. Instead, they are intended to demonstrate the kind of requests the system can handle.

User: Can you tell me how to move a file?
To move a file to another file, use mv.
For example, to move the file named paul to the file named peter,
type mv paul peter.

User: Do you know how to delete a file named dekai?
To delete the file named dekai, type rm dekai.

User: Do you know how to delete a file?
To delete a file, use rm.
For example, to delete the file named lisa, type rm lisa.

User: How can I delete the file charlie?
To delete the file named charlie, type rm charlie.

User: How can I get rid of the file named boob?
To delete the file named boob, type rm boob.

User: I want to delete a file.
To delete a file, use rm.
For example, to delete the file named jim, type rm jim.

User: How can I change the group execute permission of a file?
To change the group execute permission of a file, use chmod.
For example, to remove group execute permission from the file
named jim, type chmod g-x jim.

User: How do I change the protection of a file?
To change the permission of a file, use chmod.
For example, to remove individual read permission from the file named paul, type chmod u-r paul.

User: How can I compact a file?
To compact a file, use compact.
For example, to compact the file named peter, type compact peter.

User: How can I create a directory?
To create a directory, use mkdir.
For example, to create the directory named paul, type mkdir paul.

User: How can I find a file?
To find a file, use whereis.
For example, to find the file named charlie, type whereis charlie.

User: How can I find out how much disk space I am using?
To find out how much disk space you are using, type du.

User: How can I find out who is using the system?
To find out all the users on the system, type finger.

User: How can I find out wilensky's phone number?
To find out Wilensky's phone number, type finger wilensky.

User: Tell me how to leave.
To log out, type logout.

User: I want to change my working directory.
To connect to a directory, use cd.
For example, to connect to the directory named lisa, type cd lisa.

User: What's the best way to delete a directory?
To delete a directory, use rmdir.
For example, to delete the directory named charlie, type rmdir charlie.

User: What's the command to compact a file?
To compact a file, use compact.
For example, to compact the file named marc, type compact marc.

4. Problems

There are many technical problems yet to be resolved for each component of UC. One general problem is the integration of the components of the system. Control flows unidirectionally through UC. However, there are a number of cases in which this control structure is unsatisfactory. One such problem is the relation of language analysis and inference. Most likely, it is cognitively correct that these components function concurrently to produce an interpretation of an utterance, whereas in UC they function serially.

For example, consider the process of understanding the sentence

Do you know how to print a file on the imagen.

This utterance is syntactically ambiguous in the attachment of the prepositional phrase "on the imagen". Syntactically, this may modify "you" or "a file" as well as "print". UC does not deal with this ambiguity because one of ALANA's patterns for "print" specifically looks for "on" followed by a device. However, a more elaborate analyzer would probably not include specific information that relates this preposition to the verb, but try to related them on more general principles. In such a system, the ambiguity would be a more difficult problem.

Our current approach is to build such a system, and use a kind of marker-passing algorithm to help suggest which syntactic combination to try. For example, our knowledge about printing is such that a path between printing and a device designed for printing should be easy to find. In contrast, there would be a less obvious connection between imagen and file, or imagen and the referent of "you". This "conceptual closeness" would suggest trying to related printing and the imagen with a grammatical pattern, so the correct interpretation would be arrived at without other interpretations being tested.

Properly done, such a marker-passing scheme would effect concretion as well. For example, to arrive at the connection between printing and the imagen, it is probable that one needs to access the node for "computer-printing". Thus it seems that concretion should not be a separate inference process, but one of a number of kinds of inference that are performed by a marker-passing mechanism. We are currently attempting to reform the analyzer and the inference mechanism in the direction described.

It seems that the sort of unidirectional architecture we have employed has drawbacks elsewhere in the system. There are situations in which it seems that one component should be allowed to fail, and the failure be propagated back to another component. For example, consider processing the query

How can I edit Joe's file

Initially, the goal analyzer may make a rather literal interpretation of this request. Then the planner may fail, because the file may be protected from just such an action. It seems reasonable, however, for a consultant to suggest copying the file and editing the copy. For this to happen, control must be returned to the goal analyzer, which needs to hypothesize yet another goal underlying the goal it may have suggested initially. We are attempting to design a control structure that accommodates this flow of control.

The concretion mechanism and the goal analyzer also appear to interact in important ways. For example, consider the following question:

What does ls -v do?

This question is problematic because a "literal" response to it might be "It lists the contents of the current directory." This response is possible because there is no "-v" option to the "ls" command, and it is a characteristic of this command that it ignores options it does not recognize.

A much better response would be "There is no -v option to the ls command." To produce this response, the system must recognize that the intent of the question is something like "Tell me the conventional function of the command ls -v," and not "Tell me what actually happens when we type ls -v." One way to phrase this is that "conventional function" and "effects occurring from" are two kinds of "doing." There are certainly other kinds as well. For example, the same form may refer to the steps of a process.

Therefore, it would appear to be the job of the concretion mechanism to select the appropriate interpretation. However, it seems that the concretion mechanism cannot choose this interpretation without some knowledge of typical user goals. For example, if a user is debugging a program, it would probably be appropriate to interpret the question as referring to the steps incurred in the process rather than to the process's purpose. But reasoning about the user's goals is the job of the goal analyzer, which normally is not invoked until the concretion mechanism has completed its task.

This example illustrates the need to have more communication between the concretion mechanism and the goal analyzer. Put more strongly, the example suggests that these distinctions between language analyzer, concretion mechanism and goal analyzer are somewhat artificial. At this stage of our work, it is difficult to determine whether we simply want modules that interact more, or a more radical control structure that integrates all these functions.

There are several other more specific deficiencies of which we are aware. As we discussed previously, patterns were built into ALANA on an "as needed" basis. We are attempting to produce a more accurate language specification as we develop the inference component. Also, a mechanism for doing ellipsis, which ran in a previous version of UC, has yet to be integrated into this one.

One important deficiency of our current system is that it still doesn't participate much in real conversations. It is our intention that UC function as a consultant, and not as a front end to a data base of facts about UNIX. But our current system performs little more than this. Much of the machinery is in place, in UCEgo and PAGAN in particular, to accommodate some conversational situations. We expect much of our further development to be in this direction.

5. References

(1) Allen, J. F., Frisch, A. M. and Litman, D. J. 1982. ARGOT: the Rochester dialogue system. In *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA.

(2) Chin, D. N. 1986. User modeling in UC, the UNIX consultant. In *Proceedings of the CHI-86 Conference*, Boston, MA, April.

(3) Cox, C. A. 1986 *ALANA: Augmentable LANguage Analyzer* Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 86/283.

(4) Deering, M., Faletti, J., and Wilensky, R. 1982 *Using the PEARL AI Package* Computer Science Division, University of California, Berkeley, Memorandum No. UCB/ERL M82/19

(5) Jacobs, P. S. 1984. *PHRED: A Generator for Natural Language Interfaces* Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 84/189.

(6) Jacobs, P. S. 1985. *A Knowledge-Based Approach to Language Production.* Ph.D. thesis, University of California, Berkeley.

(7) Kaplan, S. J. 1983. Cooperative Responses from a Portable Natural Language Database Query System. In *Computational Models of Discourse*, edited by Brady and Berwick. MIT Press, Cambridge, MA.

(8) Litman, D. J. and Allen, J. F. 1984. A plan recognition model for clarification subdialogues. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Palo Alto.

(9) Luria, Marc. 1982. Dividing up the question answering process. In the *Proceedings of National Conference on Artificial Intelligence*, pp. 71-74, Pittsburgh, Pennsylvania.

(10) Luria, M. 1985. Commonsense planning in a consultant system, *Proceedings, 1985 IEEE International Conference on Systems, Man, and Cybernetics.* pp. 602-606, Tucson, Arizona.

(11) Martin, J., 1985. Knowledge acquisition through natural language dialogue, *Proceedings of the 2nd Conference on Artificial Intelligence Applications*, Miami, Florida.

(12) Norvig, P. 1983. Frame Activated Inferences in a Story Understanding Program. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 624-626.

(13) Rich, Elaine. 1979. User modeling via stereotypes. In *Cognitive Science*, Vol. 3, pp. 329-354.

(14) Wilensky, R. 1983. *Planning and Understanding: A Computational Approach to Human Reasoning.* Addison-Wesley. Reading, Mass.

(15) Wilensky, R. 1986. *Some Problems and Proposals for Knowledge Representation.* Computer Science Division, University of California, Berkeley, Report No. UCB/CSD 86/294.

(16) Wilensky, R., and Arens, Y. 1980. *A Knowledge-based Approach to Natural Language Processing.* University of California, Berkeley, Electronic Research Laboratory Memorandum No. UCB/ERL/M80/34.

(17) Wilensky, R., Arens, Y., and Chin, D. 1984. Talking to Unix in English: an overview of UC. *Communications of the Association for Computing Machinery*, June.